面向多核处理器的实时优化技术:基于独立实时域的实时优化方法

冯 华 卢 凯 王小平
(国防科学技术大学计算机学院 长沙 410073)

摘 要 多核处理器具有良好的性能功耗比,因此其在实时嵌入式系统中的应用是一种趋势。然而,现有的软件结构下,多核处理器的多核特性对实时性能的提高没有帮助;甚至,多核处理器核间的资源共享使影响程序执行时间的因素变得复杂,实时任务的最坏执行时间(Worst Case Execution Time,WCET)变得更为不可预测和难以控制。基于国产飞腾处理器研究了基于多核处理器的实时系统构建和实时性能优化,提出了"基于独立实时域的实时优化方法";通过虚拟化技术把处理器分为"实时域"和"非实时域",实时任务和非实时任务运行在不同的核心上,充分利用多核处理器各个核心,高效调度实时任务和非实时任务运行。

关键词 实时系统,多核处理器,WCET
中图法分类号 TP316.2 **文献标识码** A

Real-time Optimizing Method Based on Independent Real-time Domain: Real-time Optimizing Technology Oriented to Multi-core Processors

FENG Hua LU Kai WANG Xiao-ping
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract Multi-core processors with good performance per watt is an inevitable trend in real-time embedded systems. However, the multi-core feature of multi-core processors does no help to real-time performance improvement under the software structure currently existed, even, resource sharing between the multi-core processor cores make the impact of the program execution time factors complicated and the worst case execution time (WCET) of real-time tasks become more unpredictable and difficult to control. Based on domestic FT1000 multi-core processor, real-time system building and real-time performance optimization were researched and "real-time optimization method based on independent real-time domain" was proposed. The processor was divided into "real-time domain" and "non real-time domain" through virtualization technology. Real-time tasks and non real-time tasks running on different cores take full advantage of the cores of multi-core processor, and the real-time tasks and non-real-time tasks are efficiently scheduled to run.

Keywords Real-time system, Multi-core processors, WCET

1 引言

与传统的单核处理器相比,多核处理器在不增加功耗的同时可提供更强大的处理能力,因此其在实时嵌入式系统中的应用是必然趋势。多核处理器有不同于单核处理器的固有特性,基于多核处理器的实时系统面临一些新的问题。主要有:

1)多核处理器核间的资源共享对系统可靠性、安全性产生一定的影响;

2)多核处理器使影响程序执行时间的因素变得复杂,实时任务最坏执行时间(Worst Case Execution Time,WCET)变得更为不可预测和难以控制[3];

3)多核处理器单个核的执行能力与单核处理器没有明显不同,中断延迟、调度延迟等实时指标不会因为使用多核处理器得到提高,对有强实时性要求的应用来说,如果处理器主频低、单核性能差,即使有多个核也无法满足实时性能指标要求。

基于多核处理器的实时嵌入式系统的研究是近几年研究的热点,针对多核处理器在实时系统应用中产生的问题,国内及国际学术界、工业界从系统架构、调度策略、虚拟化、中间件等方面提出了许多解决的途径和办法。文献[1]提出一种异构多核处理器实时操作系统架构,亦即将通信总线中的多主模式引入多核操作系统架构中。文献[6]在单核处理器操作系统的基础上提出了一种主从式操作系统构架,称为 APRIX (Asymmetric Parallel Real-time Kernel)。在不同的处理器中采用不同结构和功能的操作系统内核。它的主要特点是根据处理器的功能特点采用 AMP 技术将操作系统分成主从式结构。

到稿日期:2012-12-04 返修日期:2013-03-17

冯 华(1976-),男,博士生,副研究员,CCF 会员,主要研究方向为系统软件、操作系统,E-mail:fenghua@nudt.edu.cn;卢 凯(1973-),男,博士,教授,主要研究方向为系统软件、操作系统、异构计算;王小平(1981-),男,博士,助理研究员,主要研究方向为传感器网络。

文献[7-10,12]主要从调度方面对基于多核或多处理器的实时操作系统进行了研究。Shinpei Kato 在文献[10]中提出了一种疏散任务系统的调度算法,算法基于部分分区思想,大部分任务绑定到特定的处理器核上,少数任务可以在处理器核间迁移。模拟结果显示与目前的调度算法相比,该算法有更好的调度性能,产生更少的上下文切换。

文献[15,16]针对 WCET 进行了研究。Man-Ki Yoon 在文献[15]提出了称作"可调 WCET"的新观点,设计了一种 WCET 感知的融洽 round-robin 总线调度策略和两级 Cache 分区方法,能显著降低系统整体利用率,减小影响 WCET 的不确定因素。Andre Nógueira 和 Mário Calha 在文献[16]提出了一种旨在建立高效率和可预测多处理器硬实时系统的中间件技术。它把实时任务分为两部分:需要使用复杂资源和处理复杂算法的部分(称作 payload)、使用很少资源和执行简单算法的部分(称作 wormhole),文章中这两部分运行在不同的处理器核上。如果 payload 能在规定的时间内执行完成,wormhole 就不被加载运行,如果 payload 在规定的时间内没有执行结束,wormhole 将被加载运行,以保证整个实时任务能在规定时间内执行结束。该技术描述了一种系统开发策略,并没有从根本上解决多核处理器 WCET 难以控制的问题。

飞腾 FT1000 处理器是国防科学技术大学计算机学院研制的国产高性能通用多核微处理器。针对 FT1000 在实时应用中面临的问题,本文提出了"基于独立实时域的实时优化方法"。试验结果表明,在使用特定的调度策略时,"基于独立实时域的实时优化方法"能极大提高实时性能。

2 FT1000 处理器实时性能分析

2.1 FT1000 处理器

FT1000 处理器是国防科学技术大学计算机学院研制的国产高性能通用多核微处理器,兼容 Sparc V9 指令集。处理器主频 800MHz~1000MHz,包含 8 个处理器核,每核包含 8 个硬件线程,总共 64 个线程。每个核内有独立的 16k 一级数据 cache 和 8k 一级指令 cache。全芯片共享 4MB 二级 cache。FT1000 处理器系统结构如图 1 所示。

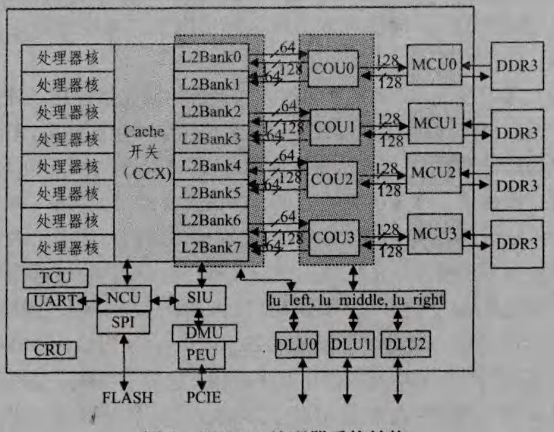

图 1 FT1000 处理器系统结构

FT1000 处理器支持硬件虚拟化,有 3 种特权级:超特权态、特权态和非特权态。系统软件结构略为复杂,包含:Hypervisor、Openboot PROM 和操作系统,其结构关系如图 2 所示。Hypervisor 运行在超特权态,为 FT1000 处理器提供虚拟化支持,它控制系统所有硬件资源[2]。外部中断首先由 Hypervisor 接收处理,然后转发给操作系统。因此,FT1000 处理器的中断处理路径较长,对实时任务来讲它的中断延迟较大,并且,Hypervisor 中锁的使用给中断转发延迟带来诸多不确定性。Openboot PROM 相当于传统的 BIOS,负责引导操作系统,搜集系统内存及外设信息,以设备树的形式提交给操作系统[1]。

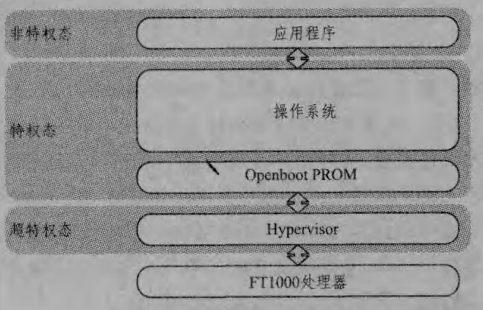

图 2 FT1000 处理器系统软件结构

2.2 FT1000 处理器实时性能分析

对 FT1000 处理器实时性能的分析主要从中断延迟、中断延迟 jitter 两个方面进行。测试平台为 8 核 64 线程 FT1000 处理器,主频 800MHz,操作系统为 Debian,Linux-2.6.35内核。这两个参数的测试方法如下。

中断延迟:以串口中断为例,在系统空闲和大负载的情况下分别测试。在串口中断服务程序中直接往该串口输出一个字符;如图 3 所示,首先在串口上输入一个字符,串口中断服务程序会在该串口输出一个字符,用示波器记录下串口上输入字符的停止位到输出字符起始位的间隔时间,即为该串口中断的处理延迟。在不同负载下分别进行 10 次测试,取平均值。使用 Stream Benchmark 测试程序作为访存负载,Stream Benchmark 是一个广泛使用的综合性访存带宽测试程序。本测试中,通过 Stream Benchmark 程序并发的线程数控制访存负载大小。

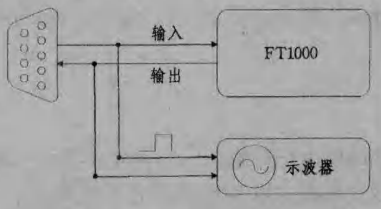

图 3 基于串口的中断延迟测试

中断延迟 jitter:同样以串口中断为例,计算不同负载下中断延迟的差异,取最大值为 jitter。

FT1000 处理器实时性能测试结果如表 1 所列。

表 1 FT1000 处理器实时性能测试结果

	latency(μs)	jitter(μs)
空载	14.86	
Stream(1 线程)	19.26	
Stream(8 线程)	29.97	66.30
Stream(16 线程)	53.23	
Stream(32 线程)	68.34	

针对 FT1000 处理器的实时性能分析数据表明,FT1000 处理器中断延迟较大,特别是在系统负载较大的时候尤其如此,中断延迟 jitter 值大。一方面是由于 FT1000 处理器系统

软件结构复杂，中断处理路径长；另一方面，FT1000 处理器主频不高，为了在单处理器包含足够多的核心，牺牲了单个核的性能，流水线简单导致串行处理能力较弱，同时访存能力受负载变化影响较大，因此在空闲和大负载时中断延迟变化大。

FT1000 处理器的系统软件以 Hypervisor 为基础，传统的基于双内核的实时优化仍然无法屏蔽 Hypervisor 对中断延迟的影响。基于以上原因，使用传统的实时优化技术很难在 FT1000 处理器上获得较好的实时性能。因此，我们提出了“基于独立实时域的实时优化方法”，并在 FT1000 处理器上进行了实现。

3 基于独立实时域的实时优化方法

FT1000 处理器包含 64 个硬件线程，在大部分嵌入式应用中很难完全用到如此多的硬件资源，因此，我们提出了“基于独立实时域的实时优化方法”。以 FT1000 处理器虚拟化技术为基础，基于虚拟机管理器 Hypervisor 把一颗 FT1000 处理器分成两个域：实时域和非实时域。非实时域运行 Linux 操作系统及非实时任务，实时域只运行实时任务。实时域和非实时域各自独立使用不同的硬件线程，由 FT1000 处理器硬件和 Hypervisor 协同进行资源和故障隔离。实时任务运行在实时域内，并且绑定到特定的硬件线程上，实时任务与非实时任务独立运行互不干扰。该方法的核心思想是实时任务调度器建立硬件线程和实时任务一对一的绑定关系，实时任务一旦加载运行即一直运行直到结束，其间不会被调度到别的线程或者停止运行。这种调度方式保证了实时任务具有最高的优先级，没有调度延迟，能够在较低性能的处理器上获得较好的实时性能。同时，独占处理器线程运行的实时任务具有更简单的运行轨迹，更容易预测和控制其 WCET。

“基于独立实时域的实时优化方法”实时域和非实时域的关系如图 4 所示。

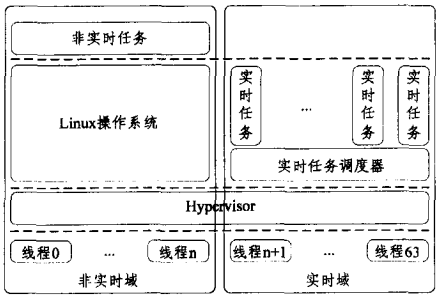

图 4 基于独立实时域的实时优化方法

基于该实时框架我们对 Linux 内核进行了修改。Linux 内核启动时使用 0～n 号处理器线程，为非实时域，$n+1$～63 号线程留给实时任务使用，为实时域。n 的值决定了实时域和非实时域所使用的处理器线程数量，通常取决于系统设计时定义的实时任务与非实时任务的负载。目前，n 的值能静态配置，不支持动态改变实时域和非实时域使用的线程数。在负载动态变化较大的情况下，存在处理器资源浪费的问题。解决该问题的方法是实现实时域和非实时域在处理器资源上的动态调整，基于 Linux 内核的 CpuHotPlug 机制，根据负载变化动态增加或者减少非实时域所使用的处理器线程数，把空闲的处理器线程添加到实时域或者把实时域未使用的线程添加到非实时域。本文发表时，该机制所涉及的负载平衡算法以及处理器线程亲和性调度策略正在设计中。为了更好地兼容于现有的实时技术，我们把 RTAI 的用户接口和通讯模块集成到了“基于独立实时域的实时优化方法”中，最大限度地保持了对 RTAI 应用的兼容性，基于 RTAI 的实时程序无需修改重新编译即可在 FT1000 平台上运行。

3.1 实时任务与非实时任务

实时任务位于实时域，运行在与 Linux 内核相同的特权态。实时任务以内核模块方式加载运行，与 Linux 内核共享地址空间，可以使用大部分不需要互斥的 Linux 内核调用。编程接口上，兼容 RTAI-3.8.1 的编程接口。非实时任务运行在非实时域的 Linux 操作系统中，是 Linux 操作系统中的普通用户进程，可以使用 Linux 操作系统的所有用户接口和库资源。

实时任务与非实时任务以及实时任务间的通讯可通过 Linux 管道或共享内存来进行。本文把 RTAI 的 SHM、FIFO 等通讯机制移植到了 FT1000 平台，最大限度地兼容了 RTAI 的程序接口。

3.2 实时任务调度器

基于独立实时域的实时优化方法主要使用简单的“一对一”调度器调度实时任务，它运行在实时域，只负责实时任务的调度运行。“一对一”调度器使实时任务与处理器线程具有一对一的绑定关系，实时任务一旦调度到某个处理器线程则一直运行。只有当实时任务数大于实时域的处理器线程数时才可能在一个处理器线程上调度多个实时任务，此时的实时任务不能独占处理器线程。实时任务可向调度器请求独占一个处理器线程，此时使用“一对一”调度策略，该实时任务可一直运行直到任务结束主动放弃线程资源。能独占处理器的实时任务数量取决于实时域分配的处理器线程数。如果处理器线程上调度了多个实时任务，则采用分时的调度策略。目前主要支持“一对一”调度策略，分时策略在基于独立实时域的实时优化方法中与其它实时技术相比没有优势，只作为备用调度策略提供。

3.3 高精度时钟

基于独立实时域的实时优化方法中，由于实时任务可以请求独占一个处理器线程，一直处于运行状态，因此，可基于处理器时钟使用查询的方式实现超高精度时钟。假设实时任务需要每隔 M 微秒调度运行一次，实时任务在每一个周期执行时间为 S 微秒，执行完成后等待 $(M-S)$ 微秒，然后进入下一个循环，如图 5 所示。该方法不需要经过漫长的中断处理流程，可以获得极高的定时精度，可以由实时任务自己控制循环等待，也可通过调度器完成。在等待期间处理器可进入休眠状态降低系统功耗。理论上讲，该方法的中断延迟可以为零，实际上如果通过调度器进行时间轮询，会有少量的开销。为了与 RTAI 接口一致，实时任务执行结束后等待下一个周期的工作我们在调度器里完成。

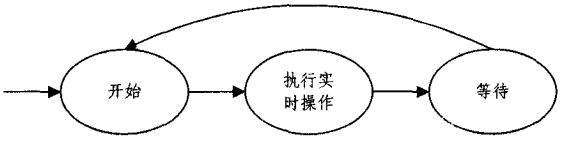

图 5 周期任务循环等待过程

3.4 实时外部中断

在基于独立实时域的实时优化方法中，实时任务对外部

中断的处理可通过轮询和系统中断的方式进行。

独占处理器线程的实时任务可屏蔽处理器中断，采用查询模式处理外部设备中断。外部设备产生的中断并不真正传递到处理器，而是由实时任务自己检测，如图 6 所示。这种方法无需经过操作系统的中断处理流程，可以获得最小的中断延迟，但实时任务需要对此特别编码实现。

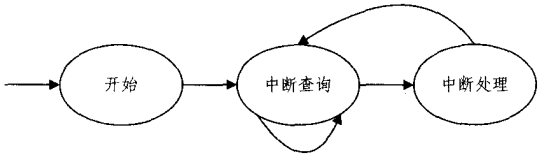

图 6　基于查询的实时外部中断处理

通过系统中断流程处理实时中断，中断仍然需要经过操作系统传递给实时任务，处理路径稍长，但实时任务可利用 OS 的中断处理接口，实现简单。

4　性能评测

我们使用 RTAI 自带测试程序集中的“latency”程序测试，对 FT1000 平台使用“基于独立实时域的实时优化方法”的实时性能进行了测试，测试在一对一调度策略下完成。“latency”测试程序是以固定时间长度为周期的循环测试程序，周期间隔设置为 125μs。测试参数 latency 为每一次调度该程序运行的延迟，取 100000 次测试结果的平均值。jitter 为 latency 参数变化幅度的最大值。测试使用 Stream Benchmark 程序作为访存负载，在不同负载下分别进行测试，测试结果如表 2 所列。

表 2　FT1000 处理器优化后的实时性能测试结果

	latency(μs)			jitter(μs)
	最小	最大	平均	
空载	0.48	2.56	0.48	
Stream(1 线程)	0.48	4.40	0.49	
Stream(8 线程)	0.48	8.00	0.52	13.22
Stream(16 线程)	0.48	12.0	0.78	
Stream(32 线程)	0.48	13.7	0.96	

作为对比，还对 Intel Pentium 2.8GHz 处理器平台使用标准 RTAI-3.8.1 的实时性能进行了测试。同样使用 RTAI 自带测试程序集中的“latency”测试程序。Intel Pentium 2.8GHz处理器与 FT1000 处理器优化前后的实时性能对比如图 7 所示，latency 参数取系统空载时的平均值。

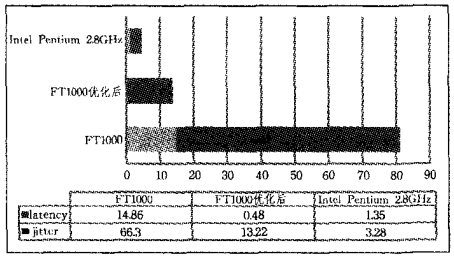

图 7　实时性能对比测试结果

从测试结果看，使用“基于独立实时域的实时优化方法”优化后的平均延迟比优化之前获得了极大的提高，甚至比 Intel Pentium 2.8GHz 处理器的延迟还小。但是延迟最大值受系统负载影响较大，分析原因主要是 Stream Benchmark 测试程序是访存密集型程序，运行 Stream Benchmark 程序的处理器核心会占用大量访存带宽从而导致运行实时任务的处理器核心访存延迟增大，影响 latency 测试结果。这也说明 FT1000 处理器的访存控制还有待改进。

“基于独立实时域的实时优化方法”简化了实时任务调度和执行轨迹，为多核处理器的实时应用提供了一个新的思路。在处理器本身性能受限的情况下，采用“基于独立实时域的实时优化方法”能极大地优化实时性能。然而，由于多核负载的影响，仍然不能保证实时任务执行的确定性，不能应用在关键的硬实时领域。

结束语　多核处理器的实时性能受核间共享资源的干扰是一直以来所面临的影响多核处理器在硬实时系统应用的主要因素。本文提出的“基于独立实时域的实时优化方法”能够在特殊的调度策略下简化实时任务执行轨迹，极大地提高实时性能。但问题依然比较严峻，而且，本文的方法是一种特殊的定制，在可扩展性以及广泛的适用性上还存在不少缺陷。目前，针对这些问题软件还没有有效的解决办法，有待处理器系统结构、系统软件等多方面的进一步研究。

参 考 文 献

[1] 蒋建春，汪同庆. 一种异构多核处理器嵌入式实时操作系统构架设计[J]. 计算机科学，2011，38(6)：298-302

[2] 冯华，卢凯. T2 处理器虚拟化技术研究[J]. 计算机工程与科学，2010，42(7)：112-117

[3] 王乐. 航空电子系统中应用多核处理器的挑战分析[J]. 航空计算技术，2011，41(5)：128-134

[4] Cedeno W，Laplante P A. an overview of real-time operating systems[J]. Journal of the Association for Laboratory Automation，2007，12：40-45

[5] Barbalace A，Luchetta A，Manduchi G，et al. Performance Comparison of VxWorks，Linux，RTAI and Xenomai in a Hard Real-time Application[J]. Nuclear Science，IEEE Transaction，55(1)：435-439

[6] Seo M，Kim H S，Maeng J C，et al. An Effective Design of Master-Slave Operating System Architecture for Multiproeessor Embedded Systems[C]//Proceedings of Lecture Notes in Computer Science，12th Asia-Padfic Conference. Seoul，Korea，2007：114-125

[7] Wei H-W，Chao Y-H，Lin S-S，et al. Current Results on EDZL Scheduling for Multiprocessor Real-time Systems[C]// 13th IEEE Internaltional Conference on Embedded and Real-time Computing Systems and Applications. Daegu，Korea，2007：114-125

[8] Block A，Brandenburg B，Anderson J H，et al. An Adaptive Framework for Multiprocessor Real-Time Systems[C]// 20th IEEE Euromicro Conference on Real-Time Systems. Prague，Czech Republic，2008：23-33

[9] Bletsas K，Andersson B. Notional processors：an approach for multiprocessor scheduling[C]// 15th IEEE Real-time and Embedded Technology and Applications Symposium. San Francisco，CA，2007：3-12

[10] Kato S，Yamasaki N，Ishikawa Y. Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors[C]// 21th Euromicro Conference on Real-time Systems. Deblin，Ireland，2009：249-258

（下转第 189 页）

的嵌套 Web Service 事务模型，并给出了其正确性描述。该模型具有很多优点，很适合开放的 Web 环境。

(1)Web Service 事务是长事务，不具有严格的原子性，整个 Web Service 事务不能等到整个事务结束才提交。该模型允许采用分阶段进行提交，在不影响业务逻辑和保证资源约束的前提下，优先提交已完成的子事务，提高了整个事务的执行效率。

(2)采用先替代后补偿的方法，当某阶段中子事务出现失败，首先考虑该子事务的替代事务，若替代事务可以成功执行，那么该阶段子事务可以顺利提交；否则，启动补偿事务对整个 Web Service 事务的失败点之前的各个阶段的子事务进行补偿，保证了整个系统的数据的完整性和一致性。

(3)为高性能地处理 Web Service 事务、灵活地维护 Web 数据的一致性以及并发地控制 Web Service 事务奠定了基础。

参 考 文 献

[1] 岳昆，王晓玲，周傲英. Web 服务核心支撑技术：研究综述[J]. 软件学报，2004，15(3)：428-442

[2] 王剑辉，吴永明. 基于细胞膜模型的 Web Service 事务管理[J]. 计算机应用与软件，2007，24(1)：87-89

[3] 郭玉彬，奚建清. Web service 的事务协调框架研究与实现[J]. 计算机工程与应用，2009，45(36)：22-25

[4] Papazoglou M P. Web 服务：原理和技术[M]. 龚玲，张云涛，译. 北京：机械工业出版社，2009：5-15

[5] Liu Cheng-fei，Zhao Xiao-hui. Towards flexible compensation for business transactions in Web service environment [J]. Service Oriented Computing and Applications，2008，2：79-91

[6] 许峰，徐碧云，黄皓，等. Web 服务事务中的补偿机制研究与实现[J]. 计算机科学，2006，33(7)：242-244

[7] 张晓雯，黄永忠，李占峻. 基于 Web Service 的工作流补偿机制[J]. 计算机工程，2009，35(24)：99-102

[8] 唐飞龙，李明禄，曹健. 一个 Web 服务事务处理模型：结构、算法和事务补偿[J]. 电子学报，2003，31(12A)：2074-2078

[9] Alrifai M，Dolog P，Balke W-T，et al. Distributed Management of Concurrent Web Service Transactions [J]. IEEE Transactions on Services Computing，2009，2(4)：289-302

[10] 申德荣，于戈，张蓉. Web 服务合成中的异构问题[J]. 东北大学学报，2004，25(3)：220-222

(上接第 162 页)

[11] Zhang Yuan-fang，Gill C，Lu Chen-yang. Real-time Performance and Middleware for Multiprocessor and Multicore Linux Platforms[C]//15th IEEE Internaltional Conference on Embedded and Real-time Computing Systems and Applications. Beijing，China，2009：437-446

[12] Baruah S. An improved global EDF schedulability test for uniform multiprocessors[C]//16th IEEE Internaltional Conference on Embedded and Real-Time Computing Systems and Applications. Macau SAR，China，2010：184-192

[13] Li Ning，Kinebuchi Y，Nakajima T. Enhancing Security of Embedded Linux on a Multi-core Processor[C]//17th IEEE Internaltional Conference on Embedded and Real-time Computing Systems and Applications. Toyama，Japan，2011：117-121

[14] Lin T-H，Kinebuchi Y，Shimada H，et al. Hardware-assisted Reliability Enhancement for Embedded Multi-core Virtualization Design[C]//17th IEEE Internaltional Conference on Embedded and Real-time Computing Systems and Applications. Toyama，Japan，2011：101-105

[15] Yoon M-K，Kim J-E，Sha Lui. Optimizing Tunable WCET with Shared Resource Allocation and Arbitrationin Hard Real-time Multicore Systems[C]//32th IEEE Real-Time Systems Symposium. Vienna，Austra，2011：227-238

[16] Nógueira A，Calha M. Predictability and efficiency in contemporary Hard RTOS for multiprocessor systems[C]//17th IEEE Internaltional Conference on Embedded and Real-time Computing Systems and Applications. Toyama，Japan，2011：3-8

(上接第 184 页)

化查询扩展算法，并在 CSE 中实现了该方法。研究的关键在于规则库中规则的形式、匹配的方式和整个扩展算法的设计。最后利用实验证明了该扩展方法比 Google 具有更好的查准率。

这里的扩展仅考虑了环境信息，为了更好地理解用户意图，生成符合用户需求的查询，未来的工作将在基于环境信息的查询扩展研究的基础上，进一步融合用户的其他上下文信息，如历史记录、用户浏览偏好以及用户社交网络等的查询扩展与查询推荐技术。

参 考 文 献

[1] Chirita P-A，Firan C S，Nejdl W. Personalized Query Expansion for the Web[C]//SIGIR 2007 Proceeding. 2007：7-14

[2] 吕碧波，赵军. 基于相关文档池建模的查询扩展[J]. 中文信息学报，2006，20(3)：78-83

[3] 王秉卿，张奇，吴立德，等. 机器学习的查询扩展在博客检索中的应用[J]. 中文信息学报，2008，22(6)：98-102，109

[4] Anderson N. Putting Search in Context：Using Dynamically-Weighted Information Fusion to Improve Search Results [C]// Eighth International Conference on Information Technology. New Generations，2011：66-71

[5] Gui Feng，Adjouadi M，Rishe N. A Contextualized and Personalized Approach for Mobile Search[C]//International Conference on Advanced Information Networking and Applications Workshops. 2009：966-971

[6] Ahmadian N，Nematbakhsh M A，Vahdat-Nejad H. A Context Aware Approach to Semantic Query Expansion[C]//International Conference on Innovations in Information Technology. 2011：57-60

[7] 陈翀，彭波，闫宏飞，等. 一种词汇共现算法及共现词对检索系统排序的影响[J]. 清华大学学报：自然科学版，2005，45(S1)：1857-1860